SETUID-FILTER METHOD FOR PROVIDING SECURE ACCESS TO A CREDENTIALS STORE FOR COMPUTER SYSTEMS

Field of the Invention

This invention relates to the field of limiting access to privileged accounts on computer systems, and, more specifically, to a method for providing controlled access to privileged accounts and sensitive data by a process that authenticates access requests, performs audit logging of the requests and retrieval of credentials of the requester in a manner that guards against hacking through the login procedure.

Background of the Invention

Computer security has become a critical issue is today's world. On one hand, enterprises are providing an ever-increasing number of services via computer systems for increasingly sophisticated, real-time transactions. At the same time, hacker break-ins, computer terrorism and employee (or former employee) sabotage is increasing. Thus, there is a tension between the need to keep computer system accessible in order to keep business moving while preventing unauthorized access.

For example, sophisticated transactions involving stocks, bonds, derivatives, futures, *etc.*, and combinations thereof, are executed internationally. These transactions are carried out on one or more secure account on one or more computer systems. Such secure accounts include, but are not limited to, trading services, price-feed services and data-feed services. These secured accounts are referred to herein as "Privileged Accounts." It is clear that Privileged Accounts must be secure to prevent tampering, unauthorized acquisition of private data, *etc.* It is also clear that that Privileged Accounts must have some form of access to keep these accounts up-to-date and operative to prevent financial loss, incorrect or missing data, unconsummated time-sensitive transactions, *etc.*

Therefore, there is a need in the art for a secure system and method for accessing Privileged Accounts.

Summary of the Invention

This problem is solved and a technical advance is achieved in the art by a system and method that provides secure access to a credential store using a setuid-filter. In accordance with this invention, a process that needs to retrieve credentials for a third party system causes the

operating system to launch a second process. This second process runs under a secured user id without interactive access using setuid facilities. The requesting process can then pass generalized command streams to the second process, including tokenized credential retrieval requests. These tokenized credential retrieval requests are processed to authenticate the requests, perform audit logging of requests and retrieval of credentials. Tokenized credential requests are transformed by the second process into credentials, which can be embedded within a command stream and then either forwarded to a sub-process or returned to the requesting process.

Brief Description of the Drawings

A more complete understanding of this invention may be obtained from a consideration of this specification taken in conjunction with the drawings, in which:

FIG. 1 is a block diagram of a data network in which an exemplary embodiment of this invention may be implemented;

FIG. 2 is a block diagram of processing according to one exemplary embodiment of this invention;

FIG. 3 is a block diagram of processing according to another exemplary embodiment of this invention; and

FIG.'s 4 - 10 are flow charts of an exemplary getpw function that implements the block diagrams of FIG. 2 and FIG. 3.

Detailed Description

In general, this invention provides a new way to secure a storage area on a computer system. Only one special user id has access to the secured storage area. An interface process changes its context so that it obtains the permissions of this special user id. The interface process remains a "black box" to the requesting user and other users. Further, there are communications channels with the interface process that also appear as "black boxes" to users. In operation, a data stream is passed through the interface process, which replaces tokenized retrieval requests in the stream with actual credentials. The interface process advantageously implements additional security measures and audit logging.

An exemplary embodiment of this invention is shown in the context of a secured password store a UNIX system using the UNIX "setuid" and "filter" facilities. Alternatives to these facilities, provided by other operating systems, allow one skilled in the art to implement this invention after studying this specification.

An infrastructure for the encrypted password store, which includes data files, is owned and only accessible by a special user id with no interactive access capability. The retrieval interface operates through a "setuid" executable that sets its user id on launch to that of the special store user id using the UNIX setuid facility. The setuid facility protects the retrieval interface program from debugging or interference or control by the requesting user. The setuid program and connected processes share a communications channel through standard UNIX inter-process pipes that cannot be read or debugged by third party processes. Retrieval requests pass, in tokenized form across a pipe that connects the requesting process with the retrieving process. The response is returned across a second pipe to the requesting process or a further processing process (depending on how the requesting process was invoked). This piping of data through the retrieving process is an application of the UNIX filter facility. The application itself may obtain encryption keys using the privileges of the special user id, perform user authorization checks on the requesting user and perform audit logging of requests and responses.

FIG. 1 is a block diagram of an exemplary data network 100 used in effecting data flow and trading of instruments. While this invention is described in terms of a data network used for financial dealings, this invention may be used wherever security is required on a computer system. For example, a server for a web site may employ this invention. Also, a stand-alone system, such as a computer-controlled telephone exchange, may employ this invention. One skilled in the art will appreciate how to implement this invention in widely diverse applications after studying this specification.

Data network 100 comprises, in general, a plurality of users, represented by servers 102, 104 and 106. Each of servers 102, 104 and 106 interact with server 108 in processing, for example, financial transactions. Further, each of servers 102, 104 and 106 communicate with server 108 using different types of processes. For example, server 102 interacts with server 108 using a "C" process 110. Server 104 interacts with server 108 by means of a "Java" process 112 and server 106 interacts with server 108 via shell scripts 114. Each server 102, 104 and 106 needs to interact with data server 120 and database 122 in performing its respective function. Examples of such functions include, but are not limited to, report generation, data feeds and database maintenance batch jobs. However, in order to maintain security of data server 120 and thus database 122, none of servers 102, 104 and 106 has a user id or a password to access database 122.

In accordance with an exemplary embodiment of this invention, processes 110, 112 and 114 communicate with a retrieval interface 126. Each of process 110, 112 and 114 pass one or more tokens to retrieval interface 126. Retrieval interface 126 is the owner of credentials store 128. Retrieval interface 126 receives the tokens from processes 110, 112 and 114 and performs a look up in credential store 128. If the tokens are recognized, then retrieval interface 126 substitutes a user id, password or other information for the tokens and returns the user id, password or other information back to the requesting process. Process 110, 112 and 114 may then access database 122 using the information delivered from retrieval interface 126. The information delivered from retrieval interface 126 is not passed back to servers 102, 104 and 106.

A management console 130 is connected to credentials store 128 in order to provide management and maintenance of the store, as is known in the art.

In the exemplary embodiment of this invention, UNIX is the operating system used in server 108. While this invention is described in terms of the UNIX operating system (and its variants), one skilled in the art will appreciate how to apply the principals of this invention to other operating systems after studying this specification.

In accordance with the exemplary embodiment of this invention, each of processes 110, 112 and 114 starts a session or a UNIX shell script that includes an invocation of the "getpw" command and the user id and password tokens for access to server120. UNIX recognizes the command "getpw" with the tokens as arguments and spawns a user process, which provides access to credential store 128. The getpw command does not have the same user id as the user's process. Thus, the user cannot invoke the getpw in the debug mode in order to access the data store.

As will be described further, below, in connection with FIG.'s 2 and 3, retrieval interface 126 takes the tokens and substitutes therefore a real user id and password. This information is passed to server 120 and the login takes place. In this manner, the user never knows the real user id and password to access server 120, but can access it none the less.

Turning now to FIG. 2, a block diagram of operation of a process in accordance with this invention is illustrated. FIG.2 illustrates a "pass-through" operation in accordance with one aspect of this invention. A user starts a shell script 202 to execute a downstream command 204 (*i.e.*, ftp, isql, *etc.*) that requires proper credentials to invoke. According to this aspect of the

invention, the process "getpw" 206 is part of a pipeline that converts a data stream. In this instance, the data stream is a short ftp script contained within an overall shell script for sata upload. Table I illustrates an exemplary shell script 202.

```
Getpw<<EOF | ftp
open server.com
user%ROLEUSER%bloomberg_ftp%
pass%ROLEPASS%bloomberg_ftp%
...
quit
EOF
```

TABLE I

The data stream 208 is passed to getpw 206 as a first filter in the pipeline. At this stage, the data stream has tokenized credentials referenced by role (*i.e.*, "%ROLEUSER%bloomberg_ftp%" and "%ROLEPASS%bloomberg_ftp%"). A "role" is also called a "command" in this exemplary embodiment. Processing proceeds into protected area 210, wherein the user does not have a user id, permissions to read, write, execute, *etc.* Getpw 206 recognizes the tokens and substitutes data retrieved from data store 212. As part of getpw 206 processing, the tokens are logged for audit purposes. New data stream 214 is passed on to the next part of the pipeline. The pipeline ends with a command 204 that takes data stream 214 as instructions. At this stage, data stream 214 has the proper credentials. Importantly, the proper credentials are only visible to the downstream command 204.

Turning now to FIG. 3, a further aspect of this invention is described in the context of a command line sequence of actions. In this exemplary embodiment, a user process 302 is invoked like a command. A data stream 304 is delivered to getpw 206 in protected area 210. getpw 206 performs its credential look up in data store 212, as described above in connections with FIG. 2, and delivers secured data back to user process 302 on data stream 306. This process may iterate. For example, user process 302 may request a user id and then a password. The user id and password are then passed from user process 302 to, for example, a database login.

While this invention is illustrated in FIG. 2 as three separate blocks, implying three separate processors, the grouping of the blocks is arbitrary. For example, shell script 202 may operate on one process and getpw 206 and downstream command 204 may operate on another. All three processes may operate on the same processor. Further, FIG. 3 is illustrated as two blocks. Blocks 302 and 206 may reside on the same or different processors, or may even be

distributed among a plurality of processors. One skilled in the art will appreciate how to structure the processing of a credential store that is advantageous to a specific application after studying this specification.

Turning now to FIG. 4, a flow chart describing processing of an exemplary embodiment of getpw 206 is shown. In this exemplary embodiment, the program described by FIG. 4 is installed as SetUID. This algorithm verifies that this program is in fact installed as SetUID, checks the protection of the directory used by the user and checks the authorization of the actual user. Error output comprises "NOAUTH" according to this exemplary embodiment.

Processing begins at oval 400 and moves first to action box 402, where the user id of the user invoking getpw 206 is obtained. Next, in action box 404, the effective user id is obtained. The effective user id comprises, in this exemplary embodiment, the user id of getpw 206. Processing continues to decision diamond 406 where a determination is made whether the user id is not the same as the effective user id. If the user id and the effective user id are the same, then getpw 306 was not properly invoked and processing proceeds to error reporting, box 408, and ends at oval 410.

If, in decision diamond 406, the user id is not the same as the effective user id, then processing proceeds to decision diamond 412. In decision diamond 412, a determination is made whether the data store directory owner is the effective user id. If it is not, then the effective user id is invalid, and processing proceeds to error reporting 408 and ends in oval 410.

If, in decision diamond 412, the data store directory owner is the effective user id, then processing proceeds to decision diamond 414, where a determination is made whether the store access permissions are exclusive to the owner. This check provides further security. If the store access permissions are not exclusive to the owner, then processing proceeds to error reporting 408 and ends in oval 410.

If, in decision diamond 414, the data store access permissions are exclusive to the owner, then processing proceeds to action box 416, where the list of authorized users is read. Processing continues to decision diamond 418, where a determination is made whether the user id is in the list of authorized users. If the user id is not in the list, processing proceeds to error reporting and ends in circle 410. If the user id is in the list, then processing proceeds through connector 420.

Turning now to FIG. 5, processing from connector 420, "filter," is shown. Getpw uses two buffers, command buffer ("cmdbuf") and token buffer ("tokbuf") as shown in Table 2.

d	command						d	token						d
%	R	O	L	E	U	..	%	b	l	o	o	m	..	%
cmdbuf							tokbuf							

TABLE 2

The following modes are used in FIG. 5 and subsequent flowcharts: FCD = find command delimiter, FTD = find token delimiter and FED = find ending delimiter.

Getpw also uses three mode variables: find command delimiter ("FCD"), find token delimiter ("FTD") and find ending delimiter ("FED"). Processing in "filter" generally initializes getpw.

Processing starts at connector 420 and moves to action box 502, where the mode is set to FCD. Processing proceeds to action box 504 where the command buffer is reset and then to action box 506, where the token buffer is reset. Processing proceeds through connector 508.

Turning now to FIG. 6, processing from connector 508, "read next," is shown. Processing starts at connector 508 and moves to action box 602, where the next character from the data stream is read. Processing continues to decision diamond 604, where a determination is made whether the character read in box 602 is an "end of file" (EOF) marker. If the character is EOF, then processing proceeds to subroutine 606 where the buffers are clears (described further, below, in connection with FIG. 9). When subroutine 606 returns, processing stops at oval 608.

If, in decision diamond 604, the character read in box 602 is not EOF, then processing continues to decision diamond 606, where a determination is made whether the mode variable is FCD. If the mode variable is not FCD, then processing proceeds to connector 608, where the buffers are filled (described below in connection with FIG. 7).

If, in decision diamond 606, the mode is FCD, then processing moves to decision diamond 610, where a determination is made whether the character read in action box 602 is a delimiter. If the character is a delimiter, then processing proceeds to action box 612, where the character is added to the command buffer. Processing continues to action box 614, where the mode is set to FTD. If, in decision diamond 610, the character is not a delimiter, then the character is written in box 616. Processing proceeds from both box 614 and box 616 to box 602.

Turning now to FIG. 7, processing at connector 608, "fill buffers," is shown. Processing starts at connector 608 and proceeds to decision diamond 702 where a determination is made whether the mode variable is FTD. If the mode variable is not FTD, then processing proceeds to connector 704, which is described further, below, in connection with FIG. 8.

If, in decision diamond 702, the mode variable is FTD, then processing continues to decision diamond 706. In decision diamond 706, a determination is made whether the character read in step 602 (FIG. 6) is a delimiter. If a delimiter is not detected, then processing proceeds to decision diamond 708, where a determination is made whether a boundary is reached. In accordance with an exemplary embodiment of this invention, a boundary comprises a line break, a buffer full or "\0." If a boundary is not reached in decision diamond 708, then processing proceeds to action box 710, where the character read in step 602 (FIG. 6) is added to the command buffer. Processing continues through connector 508 (FIG. 6, above).

If, in decision diamond 706, the character read in step 602 (FIG. 6) is a delimiter, then processing proceeds to action box 712, where the command buffer is checked. A determination is then made is decision diamond 714 where the command is valid. In accordance with an exemplary embodiment of this invention, valid commands comprise "ROLEUSER," to indicate that the token is a user id or "ROLPASS" to indicate that the token is a password.

If, in decision diamond 714, the command buffer does not have a valid command, then processing flushes the command buffer in subroutine 716 (described below in connection with FIG. 10). Processing continues to action box 718, where the character read in step 602 (FIG. 6) is added to the token buffer. If, in decision diamond 714, the command buffer has a valid command, then the character read in step 602 (FIG. 6) is added to the token buffer in box 720. Next, the mode is set to "FED."

If, in decision diamond 708, a boundary is reached, then the buffers are flushed in subroutine 606 and the character read in step 602 is written to the command buffer in box 726. In box 728, the mode is set to "FCD." Processing from boxes 718, 722 and 728 proceeds to connector 508 (FIG. 6).

Turning now to FIG. 8, processing from the FED connector 704 is shown. Processing starts at connector 704 and moves to decision diamond 802, where a determination is made whether the character read in step 602 (FIG. 6) is a delimiter. If the character is not a delimiter, then processing moves to decision diamond 804, where a determination is made whether a boundary is reached (as defined above in connection with step 708, FIG. 7). If a boundary is not reached, then the character read in step 602 (FIG. 6) is added to the token buffer in box 806.

If, in decision diamond 802, the character read in step 602 (FIG. 6) is a delimiter, then processing continues to action box 808, wherein a lookup is performed in the data store. If a

match is found in decision diamond 810, then the substitute value is written in box 812. Depending on the command that was submitted, the substituted value is a user name or a password. If a match is not found in decision diamond 810, then an error message is written. According to an exemplary embodiment of this invention, if the token was not found then "NOUSER" or "NOPASS" is returned. If there was no recognizable token, then "NOROLE" is returned.

Processing from both box 812 and 814 moves to box 816, wherein the command buffer is reset. Processing continues to box 818 where the token buffer is reset and then to box 820, where the mode is set to FCD.

If, in decision diamond 804, a boundary has been reached, then the buffers are flushed in subroutine 606 (FIG. 9) and the character read in step 602 (FIG. 6) is written in box 822. The mode is set to FCD in box 824. Processing continues from boxes 806, 820 and 824 to connector 508 (FIG. 6).

Turning now to FIG. 9, the subroutine flush buffers 606 is shown. Processing begins at connector 606 and proceeds first to the flush command buffer subroutine 716 (FIG. 10). Processing next moves to action box 902, where the token buffer is written. Next, the token buffer is reset in box 904 and the subroutine returns in oval 906.

Turning now to FIG. 10, the subroutine flush command buffer 716 is shown. Processing starts at connector 716 and moves to box 1002, where the command buffer is written. In box 1004, the command buffer is reset and the subroutine returns in oval 1006.

It is to be understood that the above-described embodiment is merely illustrative of the present invention and that many variations of the above-described embodiment can be devised by one skilled in the art without departing from the scope of the invention. It is therefore intended that such variations be included within the scope of the following claims and their equivalents.